VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Moderní rezervační systém restaurace

Modern restaurant reservation system

2014 Miroslav Berka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Miroslav Berka**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Moderní rezervační systém restaurace
Modern Restaurant Reservation System

Zásady pro vypracování:

Hlavním cílem bakalářské práce je navrhnout, implementovat a zdokumentovat webový rezervační systém pro restauraci. Při řešení postupujte podle moderních trendů pro webové aplikace. To znamená, že aplikace bude modulární, snadno rozšiřitelná a prezentační vrstva bude přizpůsobena mobilním zařízením. Cíle práce lze shrnout do těchto bodů.
1. Primární funkcí této webové aplikace bude možnost rezervace všech stolů a jídla v restauraci. Rezervaci realizujte nejlépe pomocí kalendáře, který umožní hostům zvolit datum a čas rezervace a zobrazí již obsazené termíny. Je vhodné, aby systém umožnil rezervaci celé restaurace např. pro rodinné oslavy, nebo firemní akce.
2. Dále bude webová aplikace umožňovat komunikaci s externími systémy na základě zdokumentovaného rozhraní (WebServices, WCF). Pomocí tohoto rozhraní systém může získávat informace o jídelníčcích, cenách apod. Naopak bude systém umět exportovat data o rezervovaných stolech.
3. Sekundárně by měla aplikace umožňovat správu obsahu pomocí vlastního CMS systému.
4. Z hlediska praktické implementace bude aplikace vytvářena v jazyce C# s využitím technologie ASP.NET MVC a AJAX. Pro lepší modularitu a udržitelnost aplikace bude použit Managed Extensibility Framework (MEF). Pro tvorbu databázové vrstvy vhodně použijte již existující OR Framework (.NET Entity Framework, LINQ, N-Hibernate) Je vhodné, aby byla aplikace vícejazyčná.

Seznam doporučené odborné literatury:

[1] Arlow, Jim and Neustadt, Ila. 2007. UML 2 a unifikovaný proces vývoje aplikací. s.l. : COMPUTER PRESS, 2007. 9788025115039.
[2] Freeman, Adam and Sanderson, Steven . 2011. Pro ASP.NET MVC 3 Framework, Third Edition. s.l. : APress, 2011. 978-1-4302-3405-0.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Patrik Dubec**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 7. května 2014 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2014 .....

Rád bych na tomto místě poděkoval svému vedoucímu bakalářské práce Ing. Patriku Dubcovi za podnětné připomínky a odbornou pomoc při vypracování.

Abstrakt

Hlavním cílem této bakalářské práce je návrh a implementace moderního rezervačního systému restaurace za použití ASP.NET frameworku, který bude umožňovat vytváření rezervací, jak na určité místo v restauraci, tak i na jakékoliv jídlo z jídelního lístku.
V první části bakalářské práce jsou čtenáři seznámení s použitými technologiemi.
V druhé části jsou názorné ukázky analýzy, návrhu a implementace databázové aplikace a popis vývoje.
V závěru práce jsou shrnuty veškeré poznatky s návrhy možností dalšího využití.

Klíčová slova: ASP.NET, MVC4, rezervační systém restaurace

Abstract

Main objective of this thesis is the design and implementation of a modern restaurant reservation system using the ASP.NET framework that will allow the creation of reserves, as at a certain place at a restaurant, as well as to any meal from the menu.
In first part of this thesis readers are familiar with the applied technologies.
In second part are demonstration of the design and implementation of database applications and a description of development.
The conclusion summarizes all the findings with suggestions on the possibility of further use.

Keywords: ASP.NET, MVC4, restaurant reservation system, bootstrap

Seznam použitých zkratek a symbolů

Framework	–	je softwarová struktura, která slouží jako podpora při programování, vývoji a organizace jiných softwarových projektů. Může obsahovat podpůrné programy, knihovny API, podporu pro návrhové vzory nebo doporučené postupy při vývoji.
Ajax	–	Asynchronous JavaScript and XML
CSS	–	Cascading Style Sheet
ASP.NET MVC	–	ASP.NET Model View Controller
LINQ	–	Language Integrated Query
ORM	–	Object Relational Mapping
SQL	–	Structured Query Language
API	–	Application Programming Interface
SVG	–	Scalable Vector Grapgics
HTML	–	HyperText Markup Language
WYSIWYG	–	What You See Is What You Get
CMS	–	Content management system
WCF	–	Windows Communication Foundation
MEF	–	Managed Extensibility Framework
CLR	–	Common Language Runtime

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Zadáním mé bakalářské práce je návrh, implementace a dokumentace moderního rezervačního systému pro restauraci pomocí moderních technologií. Při vývoji kladu důraz na technologii *ASP.NET MVC*, která je stále populárnější a využívanější technologií při vývoji webových aplikací.

Toto téma jsem si zvolil, protože že se zajímám a moderní webové technologie jako je *ASP.NET MVC*. Samotné téma rezervačního systému pro restauraci je do budoucna velmi praktická a výhodná součást restaurací, které tyto služby nevyužívají.

Cílem je tedy umožnit zákazníkům restaurace rezervaci jednodušším, rychlejším a přehlednějším způsobem než je tomu do teď u většiny restaurací v České republice. Kladu zde důraz na efektivní využití samotné aplikace z hlediska pozdějších úprav, jak v jídelním lístku, tak i v rozmístěním stolů v restauraci. Tento systém neměl být původně vypracován pro žádnou fyzickou osobu či společnost a měl sloužit jen k účelům bakalářské práce. Při vytváření aplikace se však objevil zájemce o systém do své restaurace, tudíž ve vývoji budu do budoucna pokračovat.

2 Použité technologie

V teoretické části bakalářské práce shrnu použité technologie při vývoji. Seznámím zde čtenáře s problematikou a význam těchto zvolených technologií.

2.1 Technologie ASP.NET

Podle knihy [1] kapitoly Úvod do ASP.NET je ASP.NET technologie firmy Microsoft pro tvorbu webových aplikacích. Je následovníkem starší technologie*ASP*, která přinesla revoluci v generování *HTML* kódu na straně serveru. Základní stavební technologií pro *ASP.NET* je *.NET Framework*. Ten poskytuje systémové služby, které podporují *ASP.NET* a přidává do operačního systému základní podpůrné služby pro *.NET* technologie.

.NET Framework obsahuje :

- společný modul pro běh programů *CLR*, kde *CLR* řídí vykonávání *.NET* kódu včetně řízení paměti a životnosti objektů v paměti. Kromě těchto řídicích služeb *CLR* umožňuje vývojářům ladit aplikace, ošetřovat chyby a využívat dědičnost mezi jednotlivými vývojovými jazyky.
- knihovny základních tříd .NET, ze kterých mohou vývojáři odvozovat své vlastní třídy. Patří sem třídy poskytující základní datové typy nebo přístupy k datům, ale také třídy, nabízející takové služby jako je kreslení, podpora síťových funkcí a mnoho dalších.

2.2 Technologie ASP.NET MVC

Společnost Microsoft představila *MVC* jako alternativu k webovým formulářům. Dle [3] je MVC architektonický vzor, jehož základní myšlenkou je oddělení logiky od výstupu. Řeší tedy problém tzv. "špagetového kódu", kdy jsou v jednom souboru (třídě) logické operace a zároveň renderování výstupu.

Podle knihy [2] kapitoly MVC je zkratka *MVC* sestavena ze tří hlavních komponent vývoje aplikace *Model - View - Controller*.

Model se odkazuje na datový model, což je něco, s čímž se provádějí základní operace nad trvalými daty aplikace, tedy vytvoření, čtení, aktualizace a odstraňování. Model je jediná část aplikace, který komunikuje s datovým skladem.

View je to, co se prezentuje uživateli - je to *ASP.NET* stránka, která využívá jeden z mnoha view enginů jako je klasický *ASPX* nebo *Razor*. Obsah stránky se bude obvykle nějak vztahovat k operacím CRUD, které se uživatel pokouší provádět.

Controler jsou soubory kódu *C#*, které tvoří mosty mezi zobrazeními s modely. Kontrolér obdrží od klienta nějaký požadavek a vybere zobrazení, které obslouží tento požadavek.

2.3 Entity Framework

Z angličtiny přeloženo [5] technologie *Entity Framework* je dalším evolučním krokem sady technologií *ADO.NET* pro přístup k datům. Technologie *Entity Framework* je ORM fra-

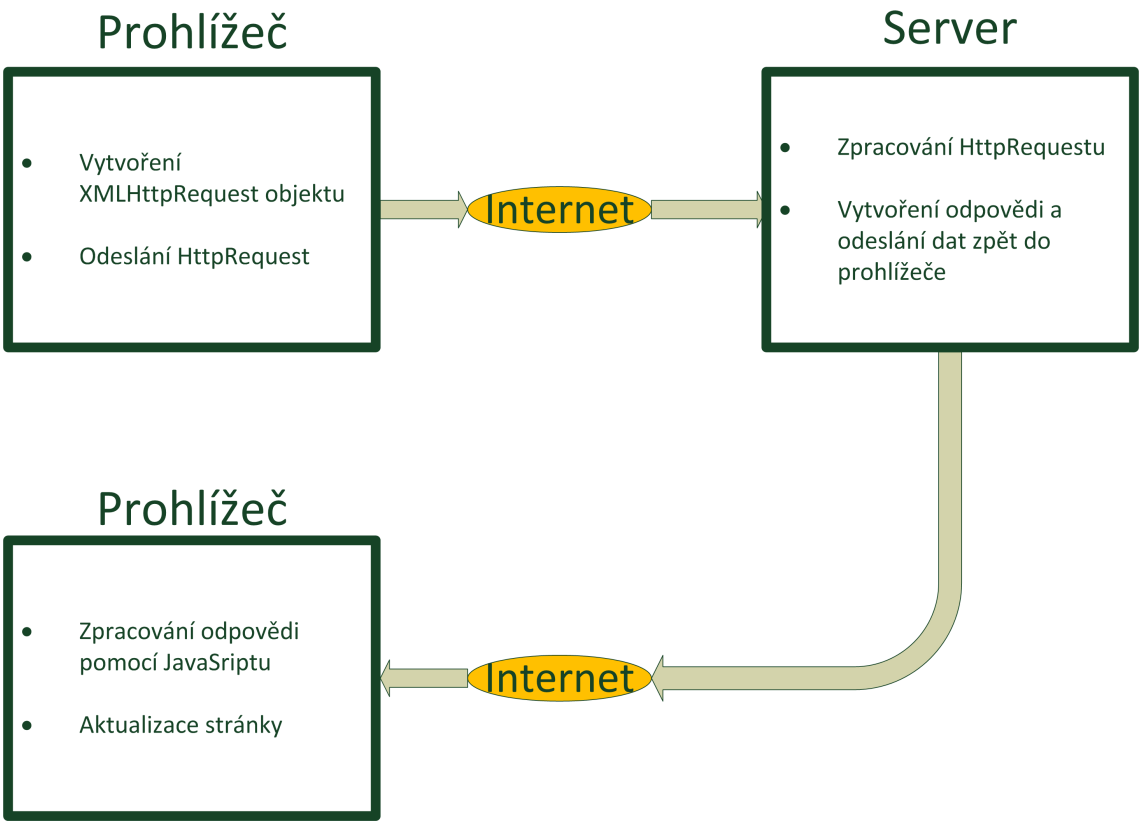
mework, který umožňuje vývojářům při programování vůči relačním databázím používat doménových modelů specifických pro aplikaci (namísto modelů specifických pro příslušné databáze). S *Entity Framework* mohou vývojáři vytvářet a udržovat datově-orientované aplikace.

2.4 C#

podle knihz [4] je to vysokoúrovňový objektově orientovaný programovací jazyk vyvinutý firmou *Microsoft* zároveň s platformou *.NET Framework*. Microsoft založil *C#* na jazycích *C++* a *Java* (a je tedy nepřímým potomkem jazyka *C*, ze kterého čerpá syntaxi). *C#* jsem si zvolil na základě osobních preferencí. Oproti skriptovacím jazykům jako např. Python, Ruby, které jsou velmi populárni pro tvorbu webových aplikací má *C#* několik výhod. Mezi ně patří např. statická typovost, mnoho standardních knihoven, výborný intellisense, analýza výkonu zabudovaná přímo v IDE a větší množství chyb je odhaleno před spuštěním kódu.

2.5 ASP.NET Ajax

Z angličtiny přeloženo dle [6] je Ajax technika pro vytváření rychlých a dynamických webových stránek. Ajax umožňuje aktualizovat části webové stránky, bez nutnosti aktualizace celé stránky. Ajax pracuje na jednoduchém principu pomocí objektu XMLHttpRequest. Princip je popsán v následujícím schématu:



Obrázek 1: Schéma principu Ajax

Typ HttpRequestu je dvojího typu a to GET nebo POST. Typ GET je přenášen v url je rychlejší a jednodušší než POST. Typ POST je přenášen v hlavičce Http Requestu a využívá se vždy, pokud je potřeba uložit data do databáze nebo se odesílá větší počet dat do databáze.

Odpověď může být typu Text, XML nebo JSON.

3 Implementace

V praktické části se zaměřím na samotnou problematiku rezervačního systému a to z hlediska pracovních postupů a implementace systému.

3.1 Zadání požadavků pro rezervační systém

V této kapitole budou specifikovány požadavky od smýšleného zákazníka (majitel restaurace).

3.1.1 Vize

Zákazník vyžaduje moderní rezervační systém pro svou restauraci. Stávající rezervační systém se řeší osobní nebo telefonickou komunikací mezi hosty a personálem. Tento způsob rezervací zpomaluje chod celé restaurace a může dojít k nepřesnostem a nedorozuměním. Systém by měl usnadnit uživatelům zarezervovat stůl, místnost a popřípadě i jídlo. Systém bude dále umožňovat rezervaci celé místnosti. Systém bude zahrnovat seznam místností se stoly, které si uživatel (zákazník restaurace) může zarezervovat a seznam všech rezervací, který bude přístupný pro zaměstnance. Systém bude dále nabízet uživateli správu svého profilu. Systém bude mít možnost nabídnout rezervaci stolů i nepřihlášeným uživatelům. Redakční systém aplikace bude umožňovat správu rezervací a jídelního lístku. Tento redakční systém bude obsluhovat administrátor.

3.2 Analýza a návrh informačního systému restaurace

V této kapitole se zabývám analýzou a návrhem informačního systému rezervace. Definuji funkční požadavky systému a na jejich základě vytvořím jednotlivé datové modely, jejich výsledkem je fyzická struktura systému.

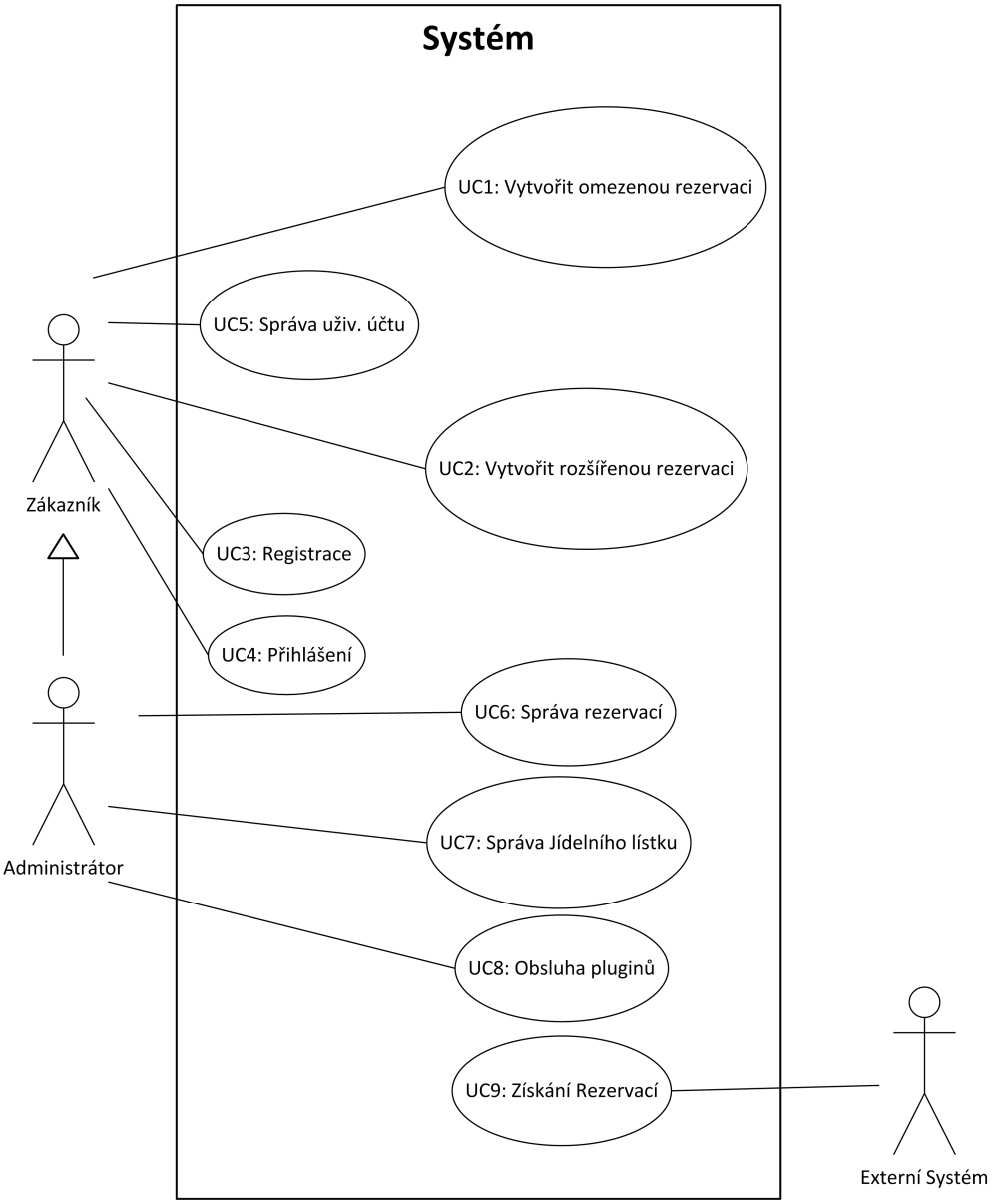
3.2.1 Funkční požadavky

Navrhovaný systém podporuje procesy spojené s vytvářením online rezervací uživatelů restaurace, pomocí online reservačního systému. rezervační systém je tvořen formou relační databáze. Funkce systému popisují následující realitu:

- FP1. Uživatel (zákazník restaurace) si vytvoří uživatelský účet, který je definovaný následujícími atributy: email, jméno, příjmení, telefonický kontakt, a heslo. Takto zaregistrovaný zákazník, může plně využívat služby rezervačního systému.
- FP2. Přihlášený uživatel může modifikovat svůj účet, například změnu telefonního čísla. Jediná nemodifikovatelná část je email, který slouží jako přihlašovací jméno.
- FP3. Nepřihlášený uživatel s omezenými službami rezervačního systému, pouze vyplní formulář s atributy: jméno, příjmení, email, telefonní kontakt, datum začátku rezervace, datum konce rezervace, požadované stoly, počet lidí a popřípadě poznámku.

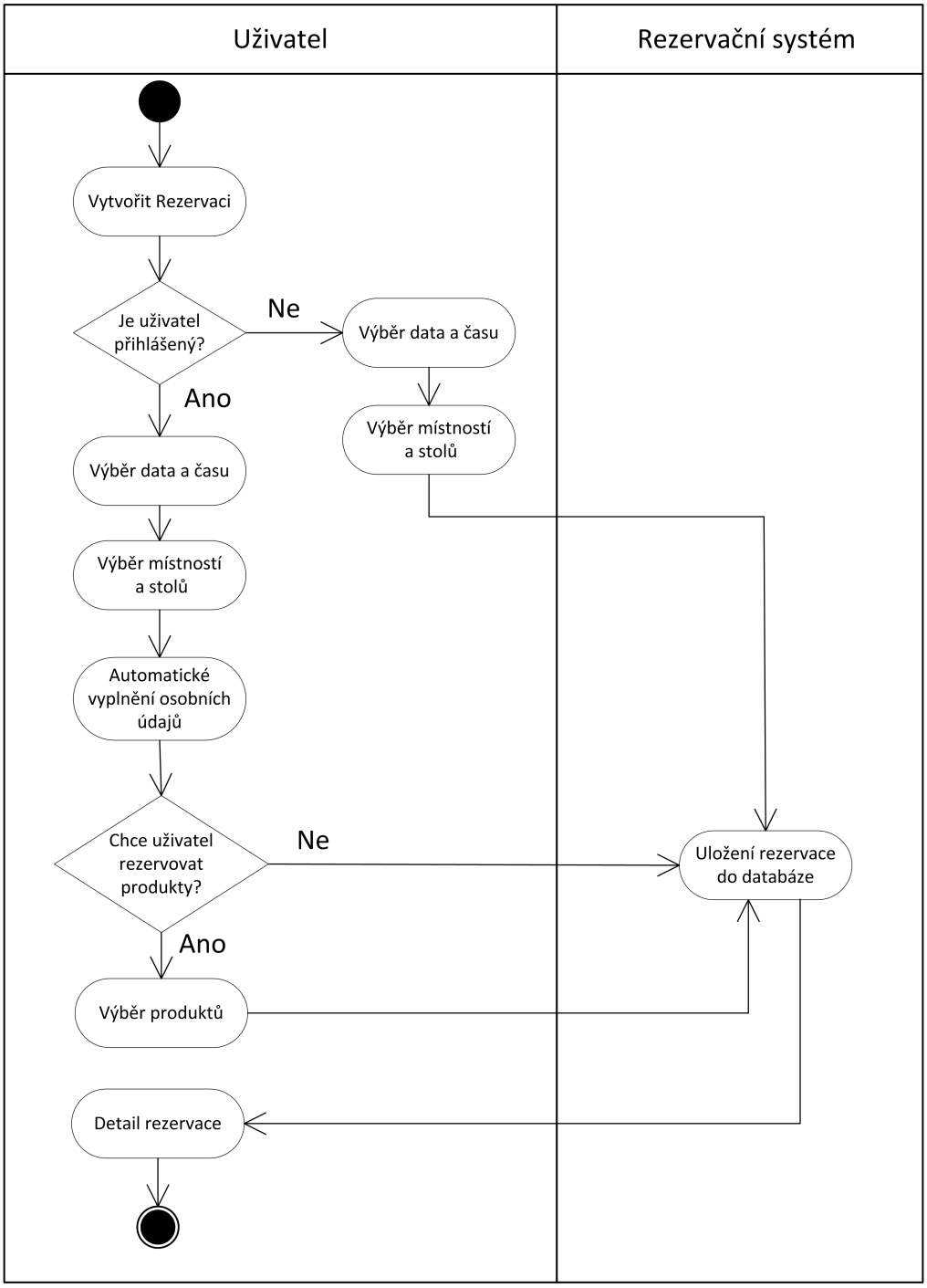
- FP4. Přihlášený uživatel, může vytvářet rezervace jak na vybrané stoly, tak i na celé místnosti, celého objektu nebo jídel. Uživatel si vytvoří novou rezervaci, která vždy obsahuje rezervaci stolu, počet lidí a datum zahájení a konce rezervace. Ostatní položky, jako rezervace jídla, nejsou nutné k rezervaci.
- FP5. Administrátor bude moct upravovat jídelní lístek a spravovat rezervace.
- FP5. Externí systém bude moct získávat výpisy rezervací.

Následující diagram případů užití zachycuje vnější pohled na modelovaný sytém. Je to dobrý a přehledný způsob, jak se se zákazníkem srozumitelně (semi-formalne) dohodnout na funkcích systému.



Obrázek 2: Diagram případů užití

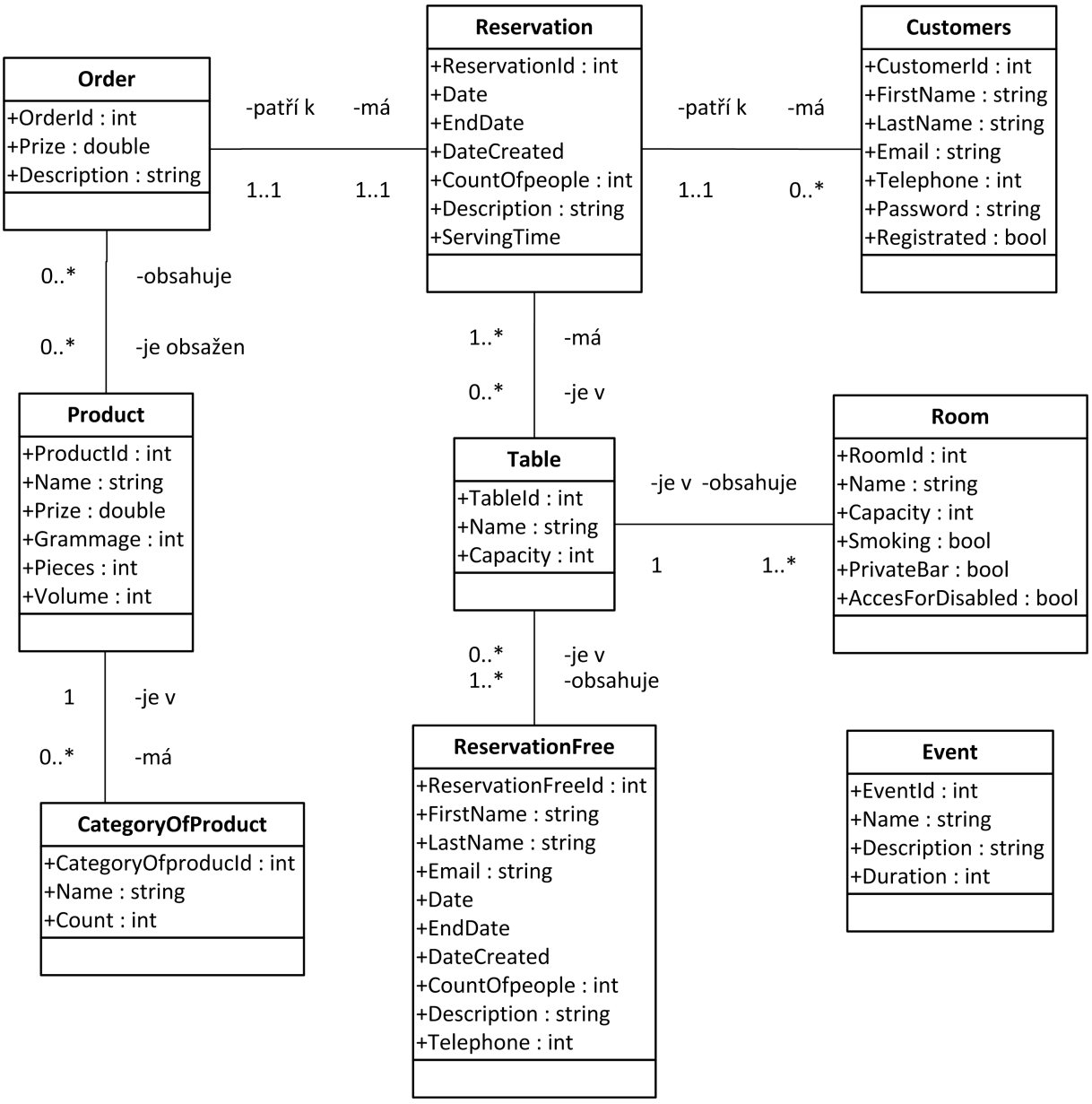
Následující diagram aktivit zachycuje hlavní proces aplikace Vytvoření rezervace. Diagram aktiv jsem zvolil, protože se díky němu dají ujasnit klíčové vlastnosti systému.



Obrázek 3: Diagram aktivit pro vytvoření rezervace

3.3 Datový Model

Použil jsem třídní diagram, který je vhodný k zachycení statické struktury systému. Třídní diagram obsahuje třídy, které jsou v databázi reprezentovány tabulkami.



Obrázek 4: Třídní diagram

Databáze je v 3. normální formě, což vede k optimálnímu využití vlastností systému při tvorbě databáze. Rozlišení rezervace mezi přihlášeným a nepřihlášeným uživatelem řeším pomocí dvou tříd. Třída Reservation je určena pro přihlášeného uživatele a třída

ReservationFree pro nepřihlášeného. Vazby mezi třídami M x N jsou tvořeny vazebními tabulkami resp. třídami, které jsou tvořeny identifikačním číslem obou tříd. Například mezi třídou Reservation a třídou Table je vazba M x N a proto vytvořím vazební třídu OrderTable.

3.4 Implementace datové vrstvy

V této části zmíním postup při vytváření databázového modulu. Pro vytváření databáze využívám *Entity Framework* a metodu *Code First*. To znamená, že vytvářím datový model, pomocí tříd, který pak *Etity Framework* převede na tabulky v databázi.

Vývoj datové části pomocí *Entity Frameworku* jsem zvolil, protože je velmi silným nástrojem pro mapování objektů, je integrovaný ve Visual Studiu 2013 a má vynikající podporu v ostatních Miscrosoft API a nástrojů, jako je WCF. Dokáže vygenerovat kompletní CRUD operace pomocí Scaffoldingu, což dokáže ušetřit mnoho času. Další velká výhoda je, že vygenerované objekty jsou pod úplnou kontrolou správce paměti (Garbage-Collector) a není tedy možné vyčerpat paměť postupnou práci s aplikacemi.

Entity Framework umožňuje připojení tříd (které jsou mapovány na tabulky) s databází velmi snadno, pomocí třídy ObjectContext a zejména třídy DbContext. Z angličtiny přeloženo [8] doporučený způsob jak pracovat s DBContextem je definovat třídu, která rozšiřuje třídu DbContext a obsahuje DBSet vlastnosti, které obsahují kolekce entit.

```
public class RestaurantContext : DbContext
    {
        public RestaurantContext()
            : base("DefaultConnection")
        {}
        public DbSet<UserProfile> UserProfiles { get; set; }
        public DbSet<Customers> Customerses { get; set; }
        public DbSet<Table> Tables { get; set; }
        public DbSet<Room> Rooms { get; set; }
        public DbSet<Order> Orders { get; set; }
        public DbSet<Product> Products { get; set; }
        public DbSet<CategoryOfProducts> CategoryOfProductses { get; set; }
        public DbSet<Reservation> Reservations { get; set; }
        public DbSet<OrderTable> OrderTables { get; set; }
        public DbSet<OrderRoom> OrderRooms { get; set; }
        public DbSet<TablesInRooms> TablesInRoomses { get; set; }
        public DbSet<OrderProduct> OrderProducts { get; set; }
        public DbSet<ReservationFree> ReservationFrees { get; set; }
        public DbSet<OrderTableFree> OrderTableFrees { get; set; }
        public DbSet<Event> Events { get; set; }
    }
```

Výpis 1: Výpis třídy RestaurantContext

Třída Reservation, která mapuje tabulku v databázi Reservation vypadá následovně:

```
public class Reservation
{
    [Key]
    public int ReservationId{ get; set;}

    [Required]
    public DateTime Date { get; set;}

    public DateTime DateCreated{ get; set;}

    [Required]
    public DateTime EndDate{ get; set;}

    [Required]
    [Range(1,10)]
    public int CountOfPeople{ get; set;}

    public string Description{ get; set;}

    public DateTime ServingTime{ get; set;}

    [Required]
    public int CustomerId{ get; set;}

    public virtual Customers Customer{ get; set;}

    public virtual Order order{ get; set;}

}
```

Výpis 2: Výpis třídy Reservation

Třída Reservation obsahuje dvě virtuální entity `Customers` a `Order`. `Virtual` označuje použití lazy loadingu. Podle článku [9] lazy loading použije identifikační klíč třídy Reservation a vytvoří instance typu proxy pro tyto dvě virtuální entity, které jsou bez dat, pouze naplněny primárními hodnotami. Ve chvíli, kdy přistoupím k třídě Reservation, načtou se všechny data z těchto entit. Výsledkem je efektivnější přenos dat mezi databází a aplikaci.

Podle studie [11] se Entity Framework nejeví jako rychlý ORM framework, oproti oblíbenému NHibernate, proto lazy loading přispívá k efektivitě. Výběr mezi ORM frameworky Entity Framework a NHibernate byl jednoduchý, jelikož jsem se o existenci NHibernate dozvěděl až v průběhu psaní této práce.

Pro průběžnou aktualizaci datového modelu (přidám nějakou vlastnost, nějakou odstraním nebo třeba jen přidám k vlastnosti atribut), musím použít tzv. *Migrace*. *Migrace* vytvořím a spustím na svém projektu pomocí příkazu `enable-migrations` v *packet manager console*.

V projektu se vytvoří nová složka `migrations`, která obsahuje iniciační migraci. Pro aktualizaci databáze, podle modelu zavolám v *packet manager console* příkaz `update-database`.

Entity Framework Code First umožňuje při inicializaci databáze rovnou naplnit tabulky daty. To je zejména užitečné při časté změně datového modelu.

`System.Data.Entity` obsahuje třídy, které slouží například k vytvoření databáze pokud žádná neexistuje, nebo smazání databáze a vytvoření nové. Tyto třídy obsahují metodu `Seed`, která by měla být přepsána a ve skutečnosti přidá data do Kontextu. V této metodě můžu tedy nastavit počáteční data jednotlivých tabulek, které vždy při inicializaci databáze naplní databázi.

```
public class CustomeresInitializer : System.Data.Entity. DropCreateDatabaseIfModelChanges<
    RestaurantContext>
     protected override void Seed(RestaurantContext context)
     {
            var customeres = new List<Customers>
                {
                new Customers()
                          {Email = "miroslav.berka@email.com",
                           FirstName = "Miroslav",
                           LastName = "Berka",
                           Telephone = "777858778",
                           Password = "123456789"},
                new Customers()
                      {Email = " bill .gates@email.com",
                       FirstName = "Bill " ,
                       LastName = "Gates",
                           Telephone = "777858779",
                           Password = "987654321"},
                };
            customeres.ForEach(c=> context.Customerses.Add(c));
            context.SaveChanges();

        }

    }
```

Výpis 3: výpis třídy CustomeresInitializer a metoda Seed

3.5 Autentizace

Autentizace je důležitou částí systému.

Podle [10] Autentizace si klade za cíl zjistit, kdo uživatel je. ASP.NET podporuje hned několik typů autentizace, například Windows nebo Passport autentizaci, ale nejčastěji používaná je Forms autentizace. Ta umožňuje vytvořit si vlastní formulář pro přihlašování (zatímco u Windows autentizace se objeví systémový přihlašovací dialog a u Passport je uživatel přesměrován na jiný web) a zadané údaje pak jsou porovnány s databází (nebo jiným úložištěm). Pokud je kombinace jména a hesla správná, můžu vytvořit cookie, která bude jakýmsi „průkazem totožnosti“ uživatele během užívání aplikace. *Forms autentizace* je použita i zde.

Pro registraci a následnou autentizaci používám `SimpleMembershipInitializer`, která je výchozím autentizačním a autorizačním mechanismem *ASP.NET MVC 4*.

Z angličtiny přeloženo [12] ve srovnání s Membership provider ASP.NET je hlavní výhoda SimpleMemebership jeho API, které je velmi jednoduché a poměrně přímočaře přebírá plnou kontrolu. Výborně se hodí pro práci s databází, která je tvořena pomocí Entity Framework Code First.

`SimpleMembershipInitializer` vytvoří v databázi své pomocné tabulky:

- **UserProfile** Tato tabulka uchovává Identifikační číslo uživatele a jeho uživatelské jméno.
- **webpages_Membership** Tato tabulka sleduje hash přihlašovací údaje uživatele a informace pro obnovení.
- **webpages_Roles** Tato tabulka má přehled o všech rolí v systému.
- **webpages_UserInRoles** Tato tabulka udržuje vazbu MxN mezi UserProfile a webpages_Roles.
- **webpages_OAuthMemebrship** Tato tabulka sleduje uživatele pomocí otevřeném ověřováním pro přihlášení do aplikace.

Z těchto automaticky vytvořených tabulek používám pouze dvě a to **UserProfile** a **webpages_Membership**.

S těmito tabulkami pracuje třída `WebSecurity`, která vytváří uživatelské účty, přihlásí a odhlásí uživatele a obnoví nebo změní heslo.

3.6 Validace

Pro spolehlivější a bezpečnější tvorbu rezervací využívám dvojí validaci a to jak na straně klienta tak na straně serveru. Výhoda validace na klientovi je ta, že klient ihned vidí, zda-li zadal nějakou chybu ve formuláři. [13] Pokud zadám neplatný vstup, zachytí ho ještě před odesláním na server validátor v JavaScriptu. Požadavek se tedy vůbec neodešle. Pro jistotu musí být ta samá validace i na serveru, protože klient si může např. JavaScript vypnout. Validace na Serveru je však důležitější. Zbraňuje vložení nebezpečných dat do databáze.

Validaci na klientské straně řeším pomocí dvou javascriptových knihoven `jquery.validate.min.js` a `jquery.validate.unobtrusive.min.js` a mírné úpravy ve `web.config`.

Nastavím tyto dva řádky na hodnotu `true`, což mi umožní dynamickou validaci na klientské straně.

```
<appSettings>
    <add key="ClientValidationEnabled" value="true" />
    <add key="UnobtrusiveJavaScriptEnabled" value="true" />
</appSettings>
```

Výpis 4: Validace ve WebConfigu

Dále je potřeba přidat tyto scripty na každou stránku určené k validaci a to v přesném pořadí:

```
<script src="/Scripts/jquery-1.7.1.js"></script>
<script src="~/Scripts/jquery.validate.min.js"></script>
<script src="~/Scripts/jquery.validate.unobtrusive.min.js"></script>
```

Výpis 5: Scripty pro Validaci

Validace využívá `DataAnnotations`, které jsou nastaveny v modelu. Datové anotace popisují pravidla pro modelové vlastnosti a *ASP.NET MVC* se bude starat o jejich uplatnění a zobrazení příslušných zpráv pro uživatele. Jsou to tedy atributy, které se definují nad třídní atributy a mohou být například typu `Required`, který znamená, že hodnota tohoto třídního atributu nesmí být prázdná.

Například pro model *Customers* a jeho atribut `FirstName` vypadají datové anotace takto:

```
public class Customers {

    [Required(ErrorMessageResourceName = "FirstNameRequired", ErrorMessageResourceType
        = typeof(AccountRes))]
      [StringLength(20, ErrorMessageResourceName = "FirstNameLength",
          ErrorMessageResourceType = typeof(AccountRes))]
      [DataType(DataType.Text, ErrorMessageResourceName = "FirstNameLengthType",
          ErrorMessageResourceType = typeof(AccountRes))]
      [LocalizedDisplayName("FirstName", NameResourceType = typeof(AccountRes))]
      public string FirstName { get; set; }
}
```

Výpis 6: Validace modelu Customers

Hodnota atributu `FirstName` musí být typu Text, vyplněna a maximálně 20 znaků dlouhá.

3.7 Uživatelské rozhraní

Podle článku [14] se přístupy z mobilních zařízení za poslední rok téměř zdvojnásobily. Porovnávaná data jsou za období listopad 2011 – říjen 2012 vs. listopad 2012 – říjen 2013. Většina z nich pochází z analytického nástroje Google Analytics. V období listopad 2011 – říjen 2012 byla návštěvnost z mobilů 1,9 % a z tabletů 1,05 %. Za poslední rok se však zvýšila na 4,32 %, respektive 2,91 % procenta. Pokud sečtu návštěvy z mobilu a tabletu, dostanu se na číslo 7,23 %. V meziročním porovnání jde o růst téměř 250 %.

Díky této studii jsem usoudil, že je více než vhodné umožnit mobilním zařízením práci s uživatelském rozhraním aplikace. Je několik způsobů jak, toho dosáhnout. Jeden ze způsobů je implementovat mobilní část aplikace zvlášť od té desktopové. Jelikož jsem se snažil využívat moderní technologie, rozhodl jsem se využít *responzivní design*, který se stává stále populárnější.

Z článku [15] je cílem responzivního designu vytvářet stránky, které uživateli poskytnou lepší pocit z jejich prohlížení. Je pro něj typická jak snadná navigace a čitelnost, tak o minimální nutnost rolování a posouvání při změně velikosti okna prohlížeče. Lze ho využít na různých přístrojích od osobních počítačů až po chytré telefony.

3.7.1 Responzivni design

Díky semináři s názvem Seminář věnovaný moderním vývojovým prostředkům a aktuálním tématům pro oblast vývoje aplikací, tentokrát vývoji v technologii *ASP.NET MVC*, který proběhl dne 21.1.2014 v Ostravě, jsem se seznámil se *frameworkem Twitter Bootstrap*.

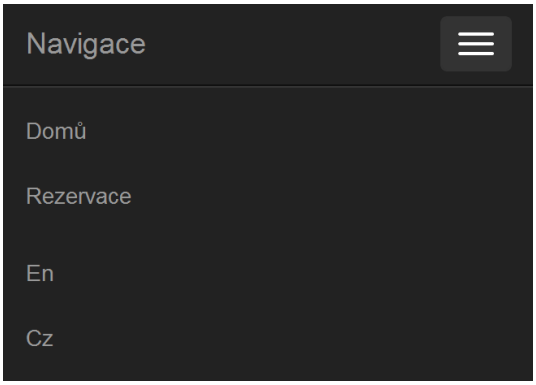
Twitter Bootstrap je velmi efektivní *framework*, který řeší velmi elegantně problematiku *responzivního designu*.

Část *bootsrapu*, kterou používám je tvořena pomocí jednoho *css* souboru. *Html* tagy pak využívají tříd, které jsou v tomto souboru definovány.

Zajímavá část je webová navigace, která se při smršťování okna prohlížeče chová tak, že při velikosti, kdy se nabídky nevejdou vedle sebe schová a vytvoří tlačítko. Po stisku tlačítka se svine paleta s nabídkami. Je to velmi efektivní zvlášť u mobilních zařízení, kde okno prohlížeče má velmi malé rozlišení.

```
<nav class="navbar navbar-fixed-top navbar-inverse" role="navigation">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".
                navbar-ex1-collapse">
                <span class="sr-only">Toggle navigation</span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <span class="navbar-brand">Navigace</span>
        </div>
        <!-- Collect the nav links, forms, and other content for toggling -->
        <div class="collapse navbar-collapse navbar-ex1-collapse">
            <ul class="nav navbar-nav">
                <li> @Html.ActionLink(@GlobalRes.MenuHome, "Index", "Home")
                </li>
                <li>@Html.ActionLink(@GlobalRes.MenuReservations, "Create", "Reservation")
                </li>
            </ul>
            <nav>
                <ul class="nav navbar-nav">
                    <li><a href="@Url.Action("changeCulture","Home",new {lang = "en"})">En
                        </a>
                    </li>
                    <li><a href="@Url.Action("changeCulture","Home",new {lang = "cs"})">Cz
                        </a>
                    </li>
                </ul>
            </nav>
        </div>
        <!-- /.navbar-collapse -->
    </div>
    <!-- /.container -->
</nav>
```

Výpis 7: Bootstrap navigace



Obrázek 5: Ukázka Bootstrap navigace

Bootsrap má svůj *Grid Systém*, což umožňuje jednotlivé části webové stránky jednoduše uspořádávat jako v tabulce.

```
<div class="container">
        <hr>
        <div class="row">

            <div class="col-lg-9 col-sm-8">
                @RenderSection("featured", required: false)
                <section class="content-wrapper main-content clear-fix">
                    @RenderBody()
                </section>
            </div>
            <div class="col-lg-3 col-sm-4">
                <section id="login">
                    @Html.Partial("_LoginPartial")
                </section>
                <hr>
            </div>
            <div id="cart"  class="col-lg-3 col-sm-4">
            </div>
        </div>
        <hr>
        <hr>
        <footer>
            <div class="row">
                <div class="col-lg-12">
                    <p>Copyright &copy; Company 2013</p>
                </div>
            </div>
        </footer>
</div>
```

Výpis 8: Bootstrap grid layout

3.7.2 Princip místností

Jedna z nejdůležitějších části rezervačního systému je rozmístění stolů v místnostech restaurace. Jelikož se rozmístění stolů v restauracích velmi často mění, například kvůli přestavbě restaurace, rozhodl jsem se umožnit správu počtu, velikosti a rozmístěni těchto stolů.

Princip, který jsem použil, má obrovskou výhodu. Je aplikovatelný na jakoukoliv restauraci. Využívám zde grafickou část *HTML5 SVG*. *SVG* pracuje s vektorovou grafikou, která lze jednoduše uložit do formátu *xml*.

Využil jsem tohoto jednoduchého principu a zkonstruoval soubor *xml*, kde jsou všechny čtverce, obdélníky , polygony a texty uloženy se svými souřadnicemi. Můžu tedy editovat rozmístění a velikosti stolů a výsledek se projeví v rezervačním formuláři. Je několik způsobů jak vytvářet obrazce. [1]

Ukázka:

```
<room1>
  <svg class="xml" width="700" height="530">
    <polygon points="0,0 0,300 140,300 140,260 130,260 130,240 140,240 140,0"
    style=" fill :rgb(168,150,150);stroke:rgb(0,0,0);stroke-width:4" />

    <rect id="3" width="60" height="60" x="370" y="130" onclick="clickMe(id)" style=" fill :rgb
        (168,150,150);stroke-width:3;stroke:rgb(0,0,0)" />

    <rect id="4" width="60" height="60" x="280" y="260" onclick="clickMe(id)" style=" fill :rgb
        (168,150,150);stroke-width:3;stroke:rgb(0,0,0)" />

    <rect id="5" width="150" height="60" x="370" y="260" onclick="clickMe(id)" style=" fill :rgb
        (168,150,150);stroke-width:3;stroke:rgb(0,0,0)" />

    <rect id="6" width="60" height="110" x="280" y="360" onclick="clickMe(id)" style=" fill :rgb
        (168,150,150);stroke-width:3;stroke:rgb(0,0,0)" />

    <text id="1" x="345" y="60" font-size = "20" fill ="black">1</text>

    <text id="2" x="300" y="175" font-size = "20" fill ="black">2</text>

    <text id="3" x="390" y="175" font-size = "20" fill ="black">3</text>

    <text x="190" y="110" font-weight = "bold" fill ="black">BAR</text>
  </svg>
</room1>
```

Výpis 9: Část XML souborů pro rozmístění stolů

[1] Využil jsem online WYSIWYG editor na stránce http://svg-edit.googlecode.com/svn-history/r1771/trunk/editor/svg-editor.html.

Obrázek 6: Ukázka Rezervace stolů

V obrázku vidíme čtyři různé barvy stolů. Barvy stolů zde ulehčují práci se systémem.

- Červená - stůl je již zarezervovaný na vybraný čas. Nejde na něm provádět žádnou akci.
- Oranžová - stůl je v budoucím čase zarezervován. Lze na tento stůl vytvořit rezervaci, avšak uživatel musí brát na vědomí, že jeho rezervace nepůjde vytvořit na jakoukoliv dobu.
- Modrá - vybraný stul uživatelem.
- Šedá (bezbarvá) - stůl je volný na jakýkoliv čas dopředu.

Pro ještě lepší práci se systémem je zde funkce pro zjištění času rezervace na jednotlivý stůl dopředu. Hodí se zejména pro uživatele, kteří chtějí vytvořit svou rezervaci na oranžových stolech. Při najetím kurzoru na stůl, vyskočí okno s výpisem rezervace dopředu.

3.7.3 Lokalizace

Aplikace má možnost zobrazit obsah v českém nebo anglickém jazyku. Výchozí jazyk je nastaven na češtinu. Lokalizace je řešena pomocí routingu, tudíž je měněna adresa url, podle toho, jaký je vybraný aktuální jazyk.

Adresy url vypadají takto:

- www.mojerestaurace.cz/cz/Home/Index.cshtml - pro češtinu.
- www.mojerestaurace.cz/en/Home/Index.cshtml - pro angličtinu.

Routovací tabulka pak vypadá následovně:

```
routes.MapRoute(
                name: "Default",
                url: "{culture}/{controller}/{action}/{id}/",
                defaults: new { culture = "cs", controller = "Home", action = "Index", id =
                    UrlParameter.Optional }
            );
```

Výpis 10: Routovací tabulka

Jednou možností jak vytvořit stránky ve více jazycích je znovu vytvořit všechny stránky ve všech jazycích. Tento způsob je pro mě velmi nepraktický, protože se musí udržovat více verzí stejného souboru. Proto jsem zvolil řešení pomocí *resources files*.

Resources file je soubor *xml*, který obsahuje řetězce, které se překládají do různých jazyků. Zdrojový kód obsahuje vždy dvojici klíč a hodnotu. Pro více jazyků, tvoříme přesně tolik souborů, kolik máme jazyků. Tyto soubory mají stejný název až na příponu, která je tvořena názvem jazyku a koncovou příponou.*Resx*.

Všechny jazykové přípony jsou obsaženy v třídě `CultureInfo`.

V době běhu *ASP.NET* využívá *resources file*, který je nejlepší pro nastavení vlastnosti `CurrentUICulture`.

Například, pokud aktuální *UI kultura* je angličtina, *ASP.NET* používá zkompilované verze souboru `AccountRes.en.resx`. V případě že není nalezen žádný soubor s touto příponou, je využíván výchozí soubor `AccountRes.resx`. Ukázka využití je ve výpisu kódu viz výše 6..

Název	Hodnota
FirstName	Jméno
FirstNameLenght	Jméno může být maximálně 20 znaků dlouhé
FirstNameLenghtType	Jméno může být pouze text
FirstNameRequired	Jméno musí být vyplněno

Tabulka 1: Část souboru AccountRes.resx

Použití zejména u Datových Anotací u modelů je jednoduché, avšak *ASP.NET* neumožňuje zobrazování názvů atributů za použití *resource files*. Datová Anotace `[Display]` uchovává tvar, jak se bude daný atribut zobrazovat. Pokud chceme mít více hodnot v

Název	Hodnota
FirstName	First name
FirstNameLenght	First name lenght is 20 char max
FirstNameLenghtType	First name is only text
FirstNameRequired	First name is required

Tabulka 2: Část souboru AccountRes.en.resx

této anotaci, musíme vytvořit třídu např. `LocalizedDisplayName`, která dědí z třídy `DisplayNameAttribute` . V modelu přidáme referenci na tuto třídu a použijeme jí. Rozdíly:

```
//before
[Display("Jmeno")]
  public string FirstName { get; set; }
//
// after
  [LocalizedDisplayName("FirstName", NameResourceType = typeof(AccountRes))]
  public string FirstName { get; set; }
```

Výpis 11: Rozdíl Datových Anotací

U principu vytváření místností viz výše 3.7.2, jsem musel využít první zmiňovanou možnost a vytvořit dva *xml* soubory ve dvou jazycích. V *javascriptové* funkci si pak určím aktuální jazyk, a podle něho zavolám název *Ajax* metody, která mi vykreslí správný *xml* soubor.

3.7.4 Výběr data při rezervaci

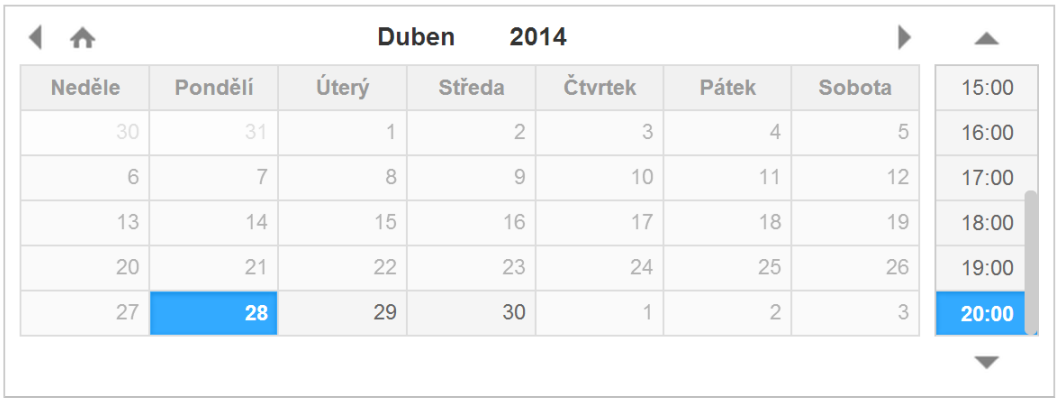
Pro lepši uživatelskou zkušenost jsem použil k volbě data kalendář, který je realizovaný *jquery* pluginem *DatetimePicker*. Tento plugin má čistý vzhled, jednoduše se s ním pracuje jak uživatelsky, tak i programátorsky. Má mnoho různých nastavení i metody, se kterými pracuji. Například pokud chci omezit časový výběr na určité hodiny, použiji následující nastavení:

```
$('#datetimepicker').datetimepicker({
      allowTimes: ['10:00', '11:00', '12:00', '13:00', '14:00', '15:00', '16:00', '17:00',
            '18:00', '19:00', '20:00'],
      });
```

Výpis 12: Nastavení hodin pro DatetimePicker

2

[2]Jquery plugin Datetimepicker je dostupný na stránce http://xdsoft.net/jqplugins/datetimepicker/



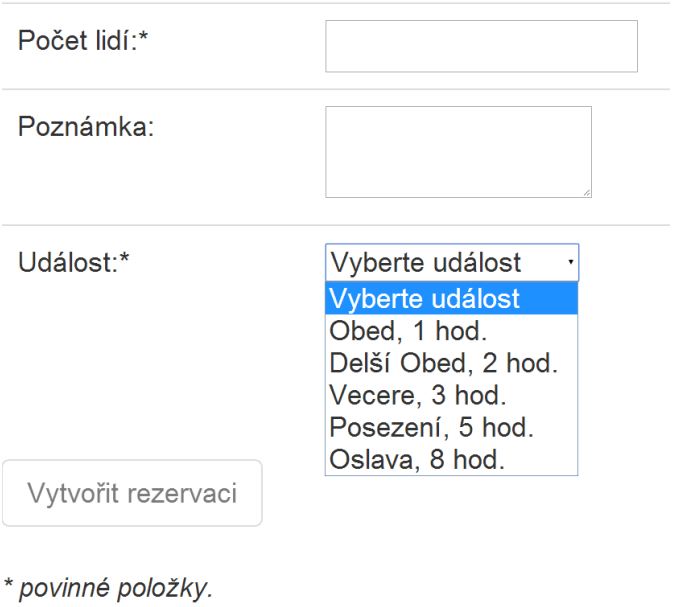
Obrázek 7: Datetimepicker

3.7.5 Vytvoření Rezervace

K vytvoření nové rezervace pro přihlášeného uživatele se postupuje následovně.

Název:	UC2 - Vytvořit rozšířenou rezervaci
Hlavní aktéři:	Uživatel
Vedlejší aktéři:	Systém
Vstupní podmínky:	Uživatel je přihlášený do systému
Hlavní scénář:	1. Uživatel zvolí datum a čas rezervace 2. Systém umožní výběr místností a stolů 3. Uživatel vybere místnost a požadované stoly 4. Systém umožní zadat doplňující informace a výběr produktů 5. Uživatel vybere produkty k rezervaci 6. Uživatel doplní zbývající informace 7. Systém provede validaci vstupních dat 8. Systém uloží rezervaci do databáze 9. Systém zobrazí uživateli detail jeho rezervace

Tabulka 3: UC2: Vytvořit rozšířenou rezervaci



Obrázek 8: Rezervační formulář

V této fázi jsem řešil několik scénářů, jak určit délku rezervace. Problémem je zde určit konec rezervace. Oproti jiným rezervačním systémům např. kino je jednoduché určit konec rezervace tak, že po skončení filmu je i konec rezervace na určenou sedačku.

V případě rezervačního systému restaurace je tento problém složitější, jelikož systém nemůže rozhodovat za uživatele, jak dlouho bude v restauraci.

První ze scénářů je spolupráce rezervačního systému společně s obsluhou restaurace. Zde by to fungovalo tak, že obsluha, která by měla nepřetržitý přístup k aplikaci, by potvrzovala rezervace, pokud by časově nezasahovaly do jiných rezervací. Tento způsob je efektivní pro restaurace, kde tolik rezervací nebývá.

Druhý způsob je omezit uživatelovou rezervaci časem, který si sám vybere jako délku rezervace. Uživatel zde musí vybrat čas začátku a konce rezervace. Zde se o celou rezervaci stará pouze rezervační systém a není potřeba zásahu obsluhy restaurace. Druhý způsob je velmi rychlý a spolehlivý. Pokud uživatel překročí čas své rezervace, může nastat problém. Uživatel se bude muset přemístit na jiný volný stůl v restauraci, nebo úplně odejít.

Vzhledem k flexibilitě systému jsem zvolil druhý ze scénářů. Uživatel při vytváření nové rezervace zvolí v posledním kroku délku rezervace v podobě přednastavených událostí. Zde se zavolá ajaxová funkce a zkontroluje uživatelský výběr, zdali nezasahuje do jiné rezervace. Pokud je vše v pořádku, uživateli se zpřístupní tlačítko pro vytvoření této rezervace. Celá tvorba rezervace je velmi jednoduchá a rychlá díky asynchronním funkcím.

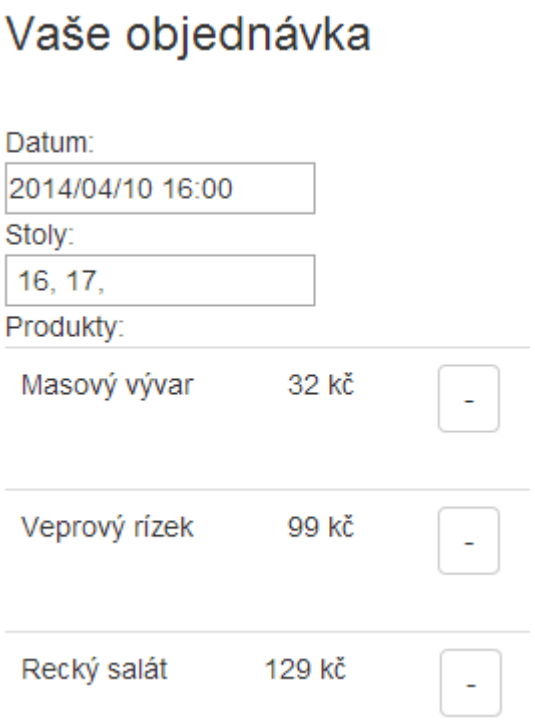
```
$(function () {
    $('#SelectedEventID').change(function (ev) {
        $(".validate").show();
        $(".loading-gif").show();
        var districtId = $(this).val(); //value of choosen eventID

        $.ajax({
            url: '/cs/Reservation/SelectEventId',    // take ActionResult SelectEventId in
                ReservationController
            type: "Get",    // methot Get
            data: { id: districtId },
            success: function (data) { // return "success" or array of taken tables
                var res = data;
                $(".loading-gif").hide();
                $(".validate").hide();
                if (res == "success") {
                    document.getElementById("btnsubmit").disabled = false;
                    $(".errorvalidate").text("");
                } else {
                    document.getElementById("btnsubmit").disabled = true;
                    $(".errorvalidate").text("Nelze rezervovat, stoly cislo: " + res + ". Zvoleny
                        termin zasahuje do jine rezervace");
                }
            }
        });
    });
});
```

Výpis 13: Ajax funkce pro validaci vstupních dat

Tento princip je založen na ajaxu a jquery. Pomocí selektoru jquery získám vybrané identifikační číslo události. Ajaxová funkce zavolá controller `Reservation` a v něm metodu *SelectEventId* s parametrem vybrané `id` události. Metoda *SelectEventId* si načte vše potřebné data jako je délka vybrané události, datum zahájení a číslo stolu a ověří uživatelovou rezervaci. Metoda vrátí odpověď typu *JSON*. Tuto odpověď ajaxová funkce dále zpracuje a zobrazí.

V pravé straně webové stránky se zobrazuje *widget*, který přehledně zobrazuje vybrané stoly, produkty a datum.



Obrázek 9: Objednávka

3.8 Redakční systém

Mým úkolem bylo vytvořit vlastní redakční systém (CMS). Do redakčního systému má vstup povolen pouze uživatel, který zná požadované heslo. Uživatelem se zde myslí zaměstnanec restaurace. V redakčním systému je možno sledovat všechny rezervace. Rezervace lze upravovat, nebo mazat.

Redakční systém by měl umožňovat editaci celého jídelního lístku. V editaci by mělo být zahrnuta možnost vytváření jídel podle určité jídelní kategorie, upravovat jednotlivým jídlům jejich vlastnosti a mazat jednotlivé jídla. Dále by měl systém umět vytvářet a upravovat denní meny. Další možností redakčního systému je obsluhovat pluginy.

Ukázka přehledu rezervací je v příloze 12

3.9 Export/Import dat

Z hlediska dalšího vývoje a možnou žádaností dat restaurace, jsem se rozhodl vytvořit univerzální rozhraní pro export/import dat o rezervovaných stolech. Toto rozhraní je tvořeno pomocí *WCF*.

Pro tento export/import dat jsem mohl využít starší technologii *webových služeb*, ale rozhodl jsem se využít novější technologii, protože je výkonnější. Hlavním rozdílem je, že

webové služby používají `XmlSerializer`, ale *WCF* používá `DataContractSerializer`, který je výkonově lepší ve srovnání s `XmlSerializer`.

WCF služba obsahuje dvě metody pro získání rezervací ze systému. Jedna z nich exportuje rezervace ze systému omezené datem, které si uživatel vybere. Druhá metoda exportuje všechny dosavadní rezervace ze systému. U druhé metody může být výstup *WCF* služby na klientské straně rozsáhlý *xml* soubor, který se s časem bude stále rozšiřovat, proto je výhodnější použít právě technologii *WCF*, která má výkonnější serializátor a deserializátor. *WCF* je navíc zpětně kompatibilní s *webovými službami*

3.10 Rozšiřitelnost aplikace

Rozšiřitelnost aplikace je velmi moderní a běžný způsob, jak obohatit svou aplikaci. Pro snadné rozšířeni jsem umožnil vytvářet a používat v aplikaci pluginy. Pluginová architektura umožňuje přidávat funkcionalitu bez zásahu do jádra aplikace, což přispívá jak k rychlejšímu vývoji, tak k vysoké stabilitě vývojových verzí aplikace. Velkou výhodou je možnost vývoje 3-ti stranou, snazší udržitelnost a rozšiřitelnost. Například aplikace bude nasazena v různých restaurací, přičemž každá z těchto restaurací využije jiné možnosti, které se těmito pluginy implementují. Tato architektura, tak nabízí možnost licencování různé vybavení verzí aplikace. Pluginová aplikace se dá vytvořit hned několika způsoby. Jedním z nich je využít populární *Managed Extensibility Framework*, který se hodí hlavně pro desktopové aplikace, nebo zkombinovat jednoduchou práci s *assembly* projektu a *PrecompiledMvcViewEngine*.

V mém případě jsem využil druhou ze zmiňovaných možností. Jelikož jsem již v minulosti tento způsob využil a je velmi jednoduchý na práci a jeho správu.

Jedním z pluginů, které jsou v aplikaci je plugin Statistika jídla. Tento plugin počítá počet objednaných jídel a zobrazuje jeho statistiku. Návod jak vytvořit plugin a importovat ho do aplikace je dodán v příloze. 6.1

4 Závěr

Mým cílem při tvorbě této práce bylo navrhnout, implementovat a zdokumentovat moderní rezervační systém restaurace. Při návrhu a implementaci systému jsem využíval poznatky, které jsem se naučil ve škole, své zkušenosti a odborné knihy.

Při návrhu databáze jsem využil Entity Framework, se kterým jsem již pracoval s minulosti. Z odborných literatur jsem se dozvěděl, že výkon Entity Frameworku je nízký oproti dnešním ostatním ORM frameworkům jako je například NHibernate. Tato nevýhoda však nehraje velkou roli, protože aplikace nepracuje s takovým množstvím dat, kde by se toto omezení projevilo.

Při tvorbě jsem kladl důraz na rozšiřitelnost a udržitelnost aplikace. Velkou výhodou této aplikace je princip rozestavění stolů v restauraci. Umožňuje dynamickou změnu velikosti a rozmístění stolů, která se následně projeví v rezervačním formuláři. Tento systém se pak dá použít pro jakoukoliv restauraci. Další důraz jsem kladl na samotnou rozšiřitelnost pomocí pluginů. Aplikace umožňuje vývoj pluginů třetí stranou bez zásahu do jádra aplikace.

Z hlediska uživatelského rozhraní jsem se snažil co nejvíce zpříjemnit tvorbu rezervací uživatelům. Rozhodl jsem se použít responzivní design. K tomuto účelu jsem využil framework Twitter Bootstrap, který vytváří přehlednou webovou stránku, jak na desktoptových, tak i na mobilních zařízeních. Uživatelské rozhraní je v českém jazyce, avšak umožňuje i přepnutí na anglickou verzi.

Aby byl systém modulární, implementoval jsem vlastní redakční systém pro správu rezervací a jídelního lístku. Tento redakční systém není dokončený. V této fázi umožňuje pouze správu rezervací, obsluhu pluginů a mírnou úpravu jídelního lístku a to mazání produktů.

Ve vývoji, zejména redakčního systému, budu dále pokračovat, jelikož chci práci zcela dokončit a nasadit na webové stránky pro skutečnou restauraci.

Vypracování této práce nebylo snadné, protože jsem se s několika technologiemi a problémy setkal poprvé. Zároveň to byl velký přínos pro můj profesní rozvoj. Dozvěděl jsem se mnoho užitečných informací, které při následujícím vývoji zcela jistě využiji. Problematice informačních systémů bych se rád i dále do budoucna věnoval.

5 Reference

[1] MacDonald, Matthew, Freeman, Adam, Szpustza, Mario, *ASP.NET 4 a C# 2010 tvorba dynamických stránek profesionálně, kniha 1*, Brno: ZONER Software, a.s, 2011.

[2] MacDonald, Matthew, Freeman, Adam, Szpustza, Mario, *ASP.NET 4 a C# 2010 tvorba dynamických stránek profesionálně, kniha 2*, Brno: ZONER Software, a.s, 2011.

[3] ČÁPKA, David. MVC Architektura. [online]. 2013, č. 1 [cit. 2014-04-28]. Dostupné z: http://www.devbook.cz/mvc-architektura-navrhovy-vzor

[4] ROBINSON, Simon a Bogdan KISZKA. *C# : programujeme profesionálně*. Brno: Computer Press, 2003. ISBN 8025100855 9788025100851.

[5] SINGH, Rahul Rajat. An Introduction to Entity Framework for Absolute Beginners. [online]. 2012, č. 1 [cit. 2014-04-28]. Dostupné z: http://www.codeproject.com/Articles/363040/An-Introduction-to-Entity-Framework-for-Absolute-B

[6] AJAX Introduction. [online]. [cit. 2014-04-28]. Dostupné z: http://www.w3schools.com/ajax/ajax_intro.asp

[7] ČINČURA, Jiří. Data/Databáze. *Entity Framework vs. LINQ to SQL* [online]. 2008, č. 1 [cit. 2014-04-15]. Dostupné z: http://www.vyvojar.cz/Articles/601-entity-framework-vs-linq-to-sql.aspx

[8] Working with DbContext. [online]. [cit. 2014-04-28]. Dostupné z: http://msdn.microsoft.com/cs-cz/data/jj729737.aspx

[9] STEIN, René. Lazy loading (zpožděné nahrávání) objektů do kolekce i ve starší aplikaci s využitím dynamické proxy. [online]. 2012, č. 1 [cit. 2014-04-29]. Dostupné z: http://blog.renestein.net/Lazy+Loading+Zpo%C5%BEd%C4%9Bneacute+Nahraacutevaacuteniacute+Objekt%C5%AF+Do+Kolekce+I+Ve+Starscaroniacute+Aplikaci+S+Vyu%C5%BEitiacutem+Dynamickeacute+Proxy.aspx

[10] KOTTNAUER, Jakub a Martin ŠIMEČEK. Seriál ASP.NET MVC. *ASP.NET MVC v praxi od A do Z: 11. díl – Autentizace a autorizace* [online]. 2009, č. 1 [cit. 2014-04-15]. Dostupné z: http://programujte.com/clanek/2009081301

[11] Collection of .NET Benchmarks ordered by most recent. [online]. 2011 [cit. 2014-04-28]. Dostupné z: https://code.google.com/p/dapper-dot-net/

[12] SINGH, Rahul Rajat. Understanding and Using Simple Membership Provider in ASP.NET MVC 4.0. [online]. 2013, č. 1 [cit. 2014-04-15]. Dostupné z: http://www.codeproject.com/Articles/689801/Understanding-and-Using-Simple-Membership-Provider

[13] ČÁPKA, David. ASP.NET MVC. *4. díl - Zpracování dat a validace v ASP.NET MVC* [online]. 2013, č. 1 [cit. 2014-04-15]. Dostupné z: http://www.devbook.cz/asp-dot-net-mvc-tutorial-zpracovani-dat-a-validace

[14] ŠIMKO, Martin. Webová analytika. *Přístupy z mobilních zařízení se za poslední rok více než zdvojnásobily* [online]. 2013, č. 1 [cit. 2014-04-23]. Dostupné z: http://webova-analytika.robertnemec.com/pristupy-mobilni-zarizeni-zdvojnasobeni/

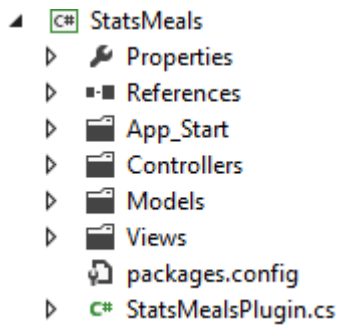
[15] SCHENKER, Marc a Miluše POKORNÁ. Co byste měli vědět o responzivním designu. [online]. 2013, č. 1 [cit. 2014-04-23]. Dostupné z: http://interval.cz/clanky/co-byste-meli-vedet-o-responzivnim-designu/

6 Přílohy

6.1 Vytvoření pluginu

Pro vytvoření Pluginu postupujte v následujících krocích:

- Ve Vaší *Solution* si vytvořte nový projekt typu *Class Library*.
- Vytvořte si stromovou strukturu jako u klasické *MVC* aplikace viz obr.



Obrázek 10: Výběr produktů

- Do vašeho projektu nainstalujte pomocí *NuGet Packages* balíček *RazorGenerator.Mvc*.
- Přidejte reference na dvě *assembly* s názvem `Infrastructure.dll` a `PluginManager.dll`, které naleznete v adresáři...
- V *root* adresáři vytvoříme třídu, která se jmenuje stejně jako projekt s postfixem *plugin*. Do této třídy přidáme referenci na `Infrastructure`. Třída bude rozšiřovat rozhraní *Iplugin* právě z tohoto *assembly*. Může tedy vypadat následovně:

```
using System;
using System.Reflection;
using Infrastructure;
namespace YourPlugin
{
    public class YourPluginPlugin : IPlugin
    {
        public string Title
        {
            get { return "Muj plugin"; }
        }
        public string Name
        {
            get { return Assembly.GetAssembly(GetType()).GetName().Name; }
        }

        public Version Version
        {
```

```
            get { return new Version(1, 0, 0, 0); }
        }

        public string EntryControllerName
        {
            get { return "YourPlugin"; }
        }
    }
}
```

Výpis 14: Rozšíření Třídy plugin rozhraním Iplugin

- V dalším kroku si pracujte jako s klasickou *MVC* aplikací, vytvořte si *modely*, *controllery* a příslušná *view*.
- Jelikož pracujeme s restaurací, budeme tedy chtít i pracovat s daty restaurace. Použijeme tedy tento `connection string`.

```
private const string ConnectionString = @"Server=b556e108-a68c-4f43-a9a8-
    a307008b347f.sqlserver.sequelizer.com;Database=
    dbb556e108a68c4f43a9a8a307008b347f;User ID=odapbqfylhwuwhmq;Password
    =
    SSFtUpnrqptiKWhtGmnJGuKGoWYuHmdtwz8ZVfAs5RuVsdwMhGoQd4gxg8f7xZQd
    ;"}
```

Výpis 15: Connection string pro databazi restaurace

- Po vytvoření *view* musíme nastavit v *properties* daného *view* vlastnost *custom tool* na hodnotu `RazorGenerator`.
- Po dokončení svého *pluginu* ho musíme zkompilovat. Výsledné *assembly* s koncovkou *.bin* zkopírujeme do hlavní aplikace do složky `plugins`.
- Aby aplikace zaregistrovala tento plugin, musíme hlavní aplikaci restartovat.
- V hlavní aplikaci se nové pluginy budou objevovat v sekci redakčního systému v navigaci. Jedinou podmínkou je, že spouštěcí *view* pluginu se musí jmenovat `Index`.

6.2 Využití WCF

Pro využívání služby pro export rezervací se musí vytvořit klientská část *WCF*. Služba je dostupná na adrese `http://localhost:/Service1.svc`. Obsahuje následující metody:

- *GetListReservations*, která vrátí list třídy `Reservationlist`. List rezervací bude obsahovat všechny rezervace.
- *GetListReservationsWithParameters*, která vrátí list třídy `Reservationlist`. Tato metoda má dva vstupní parametry `Datetime startdate` a `Datetime enddate`. Těmito parametry se omezí výpis rezervací od zadaného data po zadaný datum.

Třída `Reservationlist` vypadá následovně:

```
public class Reservationlist
{
    public int ReservationId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    public int CountOfPeople { get; set; }
    public string Telephone { get; set; }
    public string Description { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndTime { get; set; }
    public List<int> Listoftables { get; set; }
    public List<Product> Listofproduct { get; set; }
}

public class Product
{
    public int ProductID { get; set; }
    public string Name { get; set; }
    public double prize { get; set; }
    public int Gmammage { get; set; }
    public int Pieces { get; set; }
    public int Volume { get; set; }
}
```

Výpis 16: Výpis třídy Reservationlist ve WCF

Obrázek 11: Výběr produktů

Přehled rezervací

Jméno	Příjmení	E-mail	Osob	Datum zahájení	Datum ukončení	Tel. číslo	Poznámky	Upravit	Odtranit	Detail
Miroslav	Berka	Berka@email.com	5	8.4.2014 11:00:00	8.4.2014 13:00:00	777858746		Upravit	Odstranit	Detail
Mirek	Pamalsky	berrylux@seznam.cz	10	9.4.2014 18:00:00	9.4.2014 23:00:00	1234567486		Upravit	Odstranit	Detail
Miroslav	Berka	Berka@email.com	2	10.4.2014 10:00:00	10.4.2014 13:00:00	777858746		Upravit	Odstranit	Detail
Vojtěch	Šrom	sro@email.com	6	10.4.2014 11:00:00	10.4.2014 16:00:00	456123789		Upravit	Odstranit	Detail
Arnold	Berka	berrylux@seznam.cz	4	24.4.2014 18:00:00	24.4.2014 23:00:00	1234567486		Upravit	Odstranit	Detail
Tomáš	Košťálek	kosty@seznam.cz	12	30.4.2014 20:00:00	1.5.2014 4:00:00	604558775		Upravit	Odstranit	Detail
Aleš	Hemský	hama@seznam.cz	12	2.5.2014 18:00:00	2.5.2014 21:00:00	777555886		Upravit	Odstranit	Detail

Obrázek 12: Přehled rezervací